Tutoriat 11 — Exam Training

Contents

1 Exam from 2024–2025

1.1 Model 2

1.1.1 Exercise 1

Statement

What is the cost of `fork()` in the context of virtual memory with demand paging? What about `fork()` followed by `exec()`?

Write a code sequence that exemplifies the *copy-on-write* mechanism in the scenario of two processes (parent–child), and explain where the change occurs.

Solution

The cost of `fork()` is effectively *"zero"* in the context of demand paging (lazy paging), in the sense that physical memory is not copied. Instead, both the parent and the child initially point to the same physical memory frames.

The real cost comes from:

- Process Control Block (PCB) creation
- Page table creation
- Marking pages as *copy-on-write*

The cost of `fork()` followed by `exec()` is essentially the cost of `exec()`. Although `exec()` replaces the process image, demand paging ensures that memory segments are loaded only when accessed (e.g., the text segment upon instruction execution), making this operation relatively inexpensive.

```
// Random stack variable
x = 0

// Create process
fork()

// Execute read-only code (no changes)
...

// Write operation:
// A new frame is allocated to the process
// that first writes, and its page table
// entry is updated accordingly
x = 5
```

1.1.2 Exercise 3

Statement

To implement a critical section, interrupt disabling can be used. Write a short pseudo-code sequence that implements such a critical section. What are the side effects of this method and why do they occur?

Solution

```
// Non-critical code

// Enter critical section
disable_interrupts()

// Critical section code

// Exit critical section
enable_interrupts()

// Non-critical code
```

Note: You usually do not need to specify all of the following points; they are included here for completeness.

- Hardware interrupts are disabled, including the timer interrupt. This prevents time slicing, making the CPU scheduler ineffective.
- I/O operations are affected because devices (e.g., disks) cannot signal completion via interrupts.
- On multicore systems, interrupt disabling only provides mutual exclusion on the current core, not across all cores.
- If a process goes to sleep while interrupts are disabled, that CPU core becomes permanently unusable.

1.1.3 Exercise 5

Statement

Considering that segments are divided into pages of variable size, describe how you would adapt the FIFO and LRU page replacement algorithms for segments. Explain how you choose the victims and in what order. Provide a concrete example for each algorithm.

Solution

Because segments vary in size, removing a single segment may not free enough memory to allocate a new one. Therefore, for both FIFO and LRU, segments must be removed repeatedly until enough space is available.

- **FIFO:** Segments are selected as victims in arrival order: first segment, then second, and so on.
- **LRU:** Segments are removed in order of least recent usage: least recently used first, then the next least recently used, etc.

1.1.4 Exercise 6

Requirement: Consider a file system with indexing limited to a single level of indirection, containing directory-type and file-type entities. A block has 64 bytes, and a word has 16 bits.

A directory-type entity is structured as follows: at the beginning we have the directory name with a maximum of 8 characters, followed by one entry for each entity (file or directory), containing name-related information (maximum 6 characters for the name and 2 for the extension in the case of files), together with a pointer to the starting block of the file or directory.

A file-type entity consists of an index block (and the data blocks).

a. If the directory table is stored in a single block, how many files can we store at most? What is the maximum file size? Show how you designed the blocks for both cases and explain why.

b. Consider a file system with a single root directory containing the files **foo**, **bar**, and **baz** of sizes 66 bytes, 2 bytes, and 144 bytes, respectively. Write the block-level representation of this file system, including the links between blocks.

c. Modify the structure from point (b) such that **bar** is located in **/bin/bar** and **baz** is located in **/bin/lib/baz**.

Solution:

a. A block has 64 bytes. 8 bytes are used for the directory name, leaving 56 bytes. Each directory entry consumes 10 bytes (6 for the name, 2 for the extension, and 2 for a pointer to the starting block, since a word has 16 bits = 2 bytes). Thus, we can have at most $56/10 = 5.6 \implies 5$ entries, with an internal fragmentation of 6 bytes.

   Since this is a file system with a single level of indexing, there is only one **index block** containing pointers to the file's data blocks. The block size is 64 bytes; the pointer size is 2 bytes, so the index block can contain at most $64/2 = 32$ pointers. Each pointer refers to a 64-byte block, resulting in $32 \times 64 = 2048$ bytes.

b. To store the files using 64-byte data blocks, we need 2 blocks for **foo** ($64 + 2 = 66$ bytes, internal fragmentation of 62 bytes), one block for **bar** (2 bytes, internal fragmentation of 62 bytes), and three blocks for **baz** ($64 + 64 + 16 = 144$ bytes, internal fragmentation of 48 bytes).

We assume that file **foo** has a pointer to index block B_1, which contains 2 pointers to data blocks B_2 and B_3; **bar** has a pointer to index block B_4, which contains one pointer to data block B_5; and **baz** has a pointer to index block B_6, which contains 3 pointers to data blocks B_7, B_8, and B_9. The root directory contains three pointers to index blocks B_1, B_4, and B_6.

c. The **root** directory contains two entries: a file entry for **foo** (with a pointer to its index block B_1), and a directory entry for the folder **bin**. The **bin** folder contains a file entry **bar** (with a pointer to its index block B_4), and a directory entry **lib**. The **lib** folder contains a file entry **baz** (with a pointer to its index block B_6).

2 Exam from 2023–2024

2.1 Subject 2, Exercise 3

a)

- Instruction: 1 frame
- Memory operand: 1 frame

A minimum of **2 frames** is required.

b)

Thrashing occurs when a process spends more time handling page faults than executing instructions.
Example scenario:
Initial configuration:

- Process 1: frames 1 and 2
- Process 2: frame 3
- Process 3: frame 4

To execute, either Process 2 or Process 3 must steal a frame from another process. Suppose Process 2 takes a frame from Process 3. Then Process 3 faults and takes frames from Processes 1 and 2, and so on.
This leads to continuous page faults, preventing useful execution and causing thrashing.

3 (Exam from 2019-2020)

3.1 Subject 3, Exercise 1

Requirement: What is VFS? How does it help in the implementation of file systems? How do file systems coexist with its help? Give an example.

Solution: VFS stands for **"Virtual File System"** and represents an abstract interface for implementing file system functionality; it is an intermediate layer between user applications and the operating system's file systems. To implement a new file system, the abstract operations defined by VFS must be implemented (for example **read**, **write**, **open**). When an application wants to interact with the computer's file system (for example, to save or write to a file), it only needs to call the abstract functions defined in VFS, and VFS redirects the call to the correct implementation for the corresponding file system (for instance, depending on whether the file is stored on a USB drive or an HDD; these storage devices have different implementations of the functions defined by VFS).

3.2 Subject 4, Exercise 1

Requirement: Can you simulate a multi-level directory structure using a single-level structure? If yes, explain how you would implement such a simulation and how it compares to a multi-level structure. If not, explain what prevents you from doing this. Give an example.

Solution: Yes, it is possible. In a single-level structure, all files are stored in one location; to simulate multiple directories, we use file names. If we have a folder **"Documents"** that contains another folder **"personal"**, and both directories contain files, this structure can be simulated by renaming the files as follows:

- **Documents/file1.txt**
- **Documents/file2.txt**
- **Documents/personal/file1.txt**
- **Documents/personal/file2.txt**

Obviously, this method is quite unpleasant. Some disadvantages are:

- Each file must have a unique name. In a multi-level structure, files can have the same name in different directories.
- Files are stored in a linear list (or possibly using a hash table). It is inefficient to search for all files belonging to a specific folder. In a multi-level structure, a tree structure is formed.
- Renaming a folder is inefficient because many files must be renamed. In a multi-level structure, it is not necessary to rename all files, only the folder itself.